M&G No. 60001.0295US01/305243.01

REPRESENTING FIELDS IN A MARKUP LANGUAGE DOCUMENT

Related Applications

This patent application is a continuation-in-part application under 35 United States Code § 120 of United States Patent Application No. 10/187,060 filed on June 28, 2002, which is incorporated herein by reference. An exemplary schema in accordance with the present invention is disclosed beginning on page 11 in an application entitled "Mixed Content Flexibility," Serial No. _________, Docket No. 60001.0275US01, filed December 2, 2003, which is hereby incorporated by reference in its entirety.

Background of the Invention

Markup Languages have attained wide popularity in recent years. One type of markup language, Extensible Markup Language (XML), is a universal language that provides a way to identify, exchange, and process various kinds of data. For example, XML is used to create documents that can be utilized by a variety of application programs. Elements of an XML file have an associated namespace and schema.

In XML, a namespace is a unique identifier for a collection of names that are used in XML documents as element types and attribute names. The name of a namespace is commonly used to uniquely identify each class of XML document. The unique namespaces differentiate markup elements that come from different sources and happen to have the same name.

XML Schemata provide a way to describe and validate data in an XML environment. A schema states what elements and attributes are used to describe content in an XML document, where each element is allowed, what types of text contents are allowed within it and which elements can appear within which other elements. The use of schemata ensures that the document is structured in a consistent manner. Schemata may be created by a user and generally supported by an associated markup language, such as XML. By using an XML editor, the user can manipulate the XML file and generate XML documents that adhere to the schema the user has created. XML documents may be created to adhere to one or more schemata.

The XML standard is considered by many as the ASCII format of the future, due to its expected pervasiveness throughout the hi-tech industry in the coming years. Recently, some word-processors have begun producing documents that are somewhat XML compatible. For example, some documents may be parsed using an application that understands XML. However, much of the functionality available in word processor documents is not currently available for XML documents.

Summary of the Invention

The present invention is generally directed towards a method for representing an application's native field structures, such as "Creation Date of the Document", "Formula", a specially formatted number, a reference to text in another part of the document, or others in a markup language document. Fields are commonly used for document automation, so that the application itself includes certain information among the contents of the document, with possibly no extra user intervention required. The method of the invention provides a way to save this field definition information in a markup language (ML) document without data loss, while allowing the field structures to be parsed by ML-aware applications and to be read by ML programmers.

Brief Description of the Drawings

FIGURE 1 illustrates an exemplary computing device that may be used in one exemplary embodiment of the present invention;

FIGURE 2 is a block diagram illustrating an exemplary environment for practicing the present invention;

FIGURE 3 illustrates an exemplary portion of an ML file that provides representation of a simple field within ML file;

FIGURE 4 illustrates an exemplary portion of an ML file that provides representation of a complex field within ML file; and

FIGURE 5 shows an exemplary flow diagram for representing field structures in a ML document, in accordance with aspects of the invention.

Detailed Description of the Preferred Embodiment

Throughout the specification and claims, the following terms take the meanings explicitly associated herein, unless the context clearly dictates otherwise.

The terms "markup language" or "ML" refer to a language for special codes within a document that specify how parts of the document are to be interpreted by an application. In a word-processor file, the markup language specifies how the text is to be formatted or laid out, whereas in a particular customer schema, the ML tends to specify the text's meaning according to that customer's wishes (e.g., customerName, address, etc). The ML is typically supported by a word-processor and may adhere to the rules of other markup languages, such as XML, while creating further rules of its own.

The term "element" refers to the basic unit of an ML document. The element may contain attributes, other elements, text, and other building blocks for an ML document.

The term "tag" refers to a command inserted in a document that delineates elements within an ML document. Each element can have no more than two tags: the start tag and the end tag. It is possible to have an empty element (with no content) in which case one tag is allowed.

The content between the tags is considered the element's “children” (or descendants). Hence, other elements embedded in the element's content are called “child elements” or “child nodes” or the element. Text embedded directly in the content of the element is considered the element's “child text nodes”. Together, the child elements and the text within an element constitute that element's “content”.

The term "attribute" refers to an additional property set to a particular value and associated with the element. Elements may have an arbitrary number of attribute settings associated with them, including none. Attributes are used to associate additional information with an element that will not contain additional elements, or be treated as a text node.

Illustrative Operating Environment

With reference to FIGURE 1, one exemplary system for implementing the invention includes a computing device, such as computing device **100**. In a very basic configuration, computing device **100** typically includes at least one processing unit **102** and system memory **104**. Depending on the exact configuration and type of computing device, system memory **104** may be volatile (such as RAM), non-volatile (such as ROM, flash memory, etc.) or some combination of the two. System memory **104** typically includes an operating system **105**, one or more applications **106**, and may include program data **107**. In one embodiment, application **106** may include a word-processor application **120** that further includes field structures **122**. This basic configuration is illustrated in FIGURE 1 by those components within dashed line **108**.

Computing device **100** may have additional features or functionality. For example, computing device **100** may also include additional data storage devices (removable and/or non-removable) such as, for example, magnetic disks, optical disks, or tape. Such additional storage is illustrated in FIGURE 1 by removable storage **109** and non-removable storage **110**. Computer storage media may include volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information, such as computer readable instructions, data structures, program modules, or other data. System memory **104**, removable storage **109** and non-removable storage **110** are all examples of computer storage media. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computing device **100**. Any such computer storage media may be part of device **100**. Computing device **100** may also have input device(s) **112** such as keyboard, mouse, pen, voice input device, touch input device, etc.

Output device(s) **114** such as a display, speakers, printer, etc. may also be included. These devices are well know in the art and need not be discussed at length here.

Computing device **100** may also contain communication connections **116** that allow the device to communicate with other computing devices **118**, such as over a network. Communication connection **116** is one example of communication media. Communication media may typically be embodied by computer readable instructions, data structures, program modules, or other data in a modulated data signal, such as a carrier wave or other transport mechanism, and includes any information delivery media.. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. The term computer readable media as used herein includes both storage media and communication media.

Generally, the present invention is directed at representing field structures in an ML document. The ML document may be read by applications that do not share the same schema that created the document. The application not sharing the same schema may parse the field structures, regardless of whether or not the fields are understood.

FIGURE 2 is a block diagram illustrating an exemplary environment for practicing the present invention. The exemplary environment shown in FIGURE 2 is a word-processor environment **200** that includes word-processor **120**, ML file **210**, ML Schema **215**, and ML validation engine **225**.

In one embodiment, word-processor **120** has its own namespace or namespaces and a schema, or a set of schemas, that is defined for use with documents associated with word-processor **120**. The set of tags and attributes defined by the schema for word-processor **120** define the format of a document to such an extent that it is referred to as its own native ML. Word-processor **120** internally validates ML file **210**. When validated, the ML elements are examined as to whether they conform to the ML schema **215**. A schema states what tags and attributes are used to describe content in an ML document, where each tag is allowed, and which tags can appear within other tags, ensuring that the documentation is structured the same way. Accordingly, ML **210** is valid when structured as set forth in arbitrary ML schema **215**.

ML validation engine **225** operates similarly to other available validation engines for ML documents. ML validation engine **225** evaluates ML that is in the format of the ML validation engine **225**. For example, XML elements are forwarded to an XML validation engine. In one embodiment, a greater number of validation engines may be associated with word-processor **120** for validating a greater number of ML formats.

Representing Fields in a Markup Language Document

The present invention generally provides a method to represent an application's native field structures in markup language (ML) such as XML. The field structures may be parsed by applications that understand the markup other than the application that generated the ML file. Fields are commonly used for document automation, so that the application itself includes certain information among the contents of the document, with possibly no extra user intervention required. Fields may be a very powerful feature making the document authoring or editing process much more efficient.

Fields are elements of the content of a document, whose purpose is to automatically generate or modify the content, or its appearance, depending on various conditions and/or settings specified by the user. Fields may be very simple or very complex.

A defining characteristic of a field is that it is updatable. For example, a "LastSavedBy" field may insert the name of the last person who saved the document at the location of the field. When a different person saves the document from the one who saved it last time, the name inserted by the field is automatically replaced with the name of the latest user. The field therefore generates and modifies the content of the document depending the identity of the person saving the document.

A “Ref” field (reference) is a more complex example. The field’s result is text which is a “linked” copy of text from another place of the document, identified by a named bookmark. As soon as the original text changes, the text inserted by the field changes as well. The “ref” field may also affect the formatting of the copied text (e.g., by making the copied text uppercased).

An even more complex example is a field which creates a table of contents for the document by: reproducing all the headings used in the document in a single location; organizing the headings according to their level to expose the hierarchy of the document; changing the formatting of the headings; automatically including the correct page number with each heading in the table of contents; and determining the numbering style to use for the table of contents. A table of contents that is the result of such a field is automatically updatable and self-organizing based on the contents of the document. Therefore, the maintenance of a table of contents is automated so that the user is not required to create and maintain the table of contents manually.

Certain fields may refer to one another. For example, a field whose result is the Index section of a document relies on the existence of fields throughout the document that mark index entries. Also, certain fields may be nested one inside of another and work together in a “recursive” manner to create the desired result.

In order for an application to support the concept of fields, the application represents each field internally by a structure mirroring field properties. A field structure generally consists of the following two major parts:

1. field instructions
2. field result

"Field instructions" comprise the portion of a field containing pieces of information such as:

1. the name of the field;
2. zero or more arguments on which the field operates (e.g., file names, style names, bookmark names, numbers, literal text, and others); and

3. zero or more options specific to the field that further modify the behavior of the field (e.g., formatting options, numbering style settings, and others).

The "field result" comprises the portion of the field which contains the result of the operation performed by the field. The field result may simply be a number, but also may be as arbitrarily rich and complex as a whole fully formatted document or OLE (Object Linking and Embedding) object. The result is the part that is updated by the field when the value of the arguments of the field changes.

Since a field itself is an editable part of a document, it coexists with the surrounding content. The field may be separated from the surrounding content by a field start and field end marks. Also, the instructions are separated from the result. In a first embodiment, the separators are visible to the user. In a second embodiment, the separators are not visible to the user. Correspondingly, in other embodiments, the instructions may or may not be visible to the user. Typically, a user is able to choose between a view where only field instructions are visible and one where only field results are displayed.

Based on how the instructions portion of a field is structured, fields are divided into two major categories:

simple fields – The instructions portion only contains instructions, and not richly formatted content or other embedded fields.

complex fields – The instructions portion contains richly formatted content or other embedded fields.

The present invention provides a method for saving all the field information described above as ML without losing any data, by mapping the application's internal field structures described above to saved ML markup.

The present invention represents the fields in ML depending on whether the field is a "complex" field or a "simple" one. FIGURE 3 illustrates an exemplary portion of an ML file that provides representation of a simple field within ML file, in accordance with aspects of the present invention.

In the example shown, the simple field is represented by fldSimple element **310** containing instructions **320** and result **330**. Instructions **310** of the field are written out as the string value of the instr attribute. Result **330** of the field is arbitrarily rich ML content written out as the child of fldSimple element **310**. In the example given, the ML markup represents an "Author" field, whose function is to insert the name of the document author (John Doe) into the document, in upper case. Other field instructions and results may be used within a simple field, and a simple field may correspond to elements other than the fldSimple element without departing from the scope of the present invention.

FIGURE 4 illustrates an exemplary portion of an ML file that provides representation of a complex field within ML file, in accordance with aspects of the present invention. The example includes the presence of arbitrarily rich ML markup inside and around the instructions **440** and the result **450** The field's ML markup can co-exist and be intertwined with other ML markup, as shown in FIGURE 4.

As shown, instructions **410** of a complex field themselves may contain arbitrarily rich content, including other fields. Accordingly, ML for a complex field includes the definition of two empty elements such as fldChar **410** and instrText **420**. Element fldChar **410** marks the beginning of the field, the boundary between the instructions and the result, or the end of the field, depending on the value of its fldCharType attribute **430** (e.g., "begin", "separate", "end", etc.). Element instrText **420** contains the ML markup for the arbitrarily rich instructions of the field.

In one embodiment, the elements appear in the following specific order for the field representation to be valid:

```
<fldChar fldCharType="begin"/>
...
<instrText>
        Field instructions go here...
</instrText>
...
```

```
<fldChar fldCharType="separate"/>
        Field result goes here...
<fldChar fldCharType="end"/>
```

The actual contents of the field instructions may vary from application to application, depending on the types of fields the application supports.

The following is an exemplary portion of schema for generating the fields, in accordance with aspects of the present invention:

Simple Fields

```
<xsd:element name="fldSimple" type="simpleFieldType" minOccurs="0"
maxOccurs="unbounded">
        <xsd:annotation>
                <xsd:documentation>Represents a simple Word field (with plain text
instructions). Simple fields are run-time calculated entities in Word (for example, page
numbers).</xsd:documentation>
        </xsd:annotation>
</xsd:element>

<xsd:complexType name="simpleFieldType">
        <xsd:annotation>
                <xsd:documentation>Defines a field in the
document.</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
                <xsd:element name="fldData" type="stringType" minOccurs="0"
maxOccurs="1">
                        <xsd:annotation>
                                <xsd:documentation>Represents field
data.</xsd:documentation>
                        </xsd:annotation>
                </xsd:element>
                <xsd:choice minOccurs="1" maxOccurs="unbounded">
                        <xsd:element name="r" type="rElt">
                                <xsd:annotation>
                                        <xsd:documentation>Represents the run element.
This is the leaf container for data in a Word document -- text, pictures, and so
on.</xsd:documentation>
                                </xsd:annotation>
                        </xsd:element>
                        <xsd:element name="fldSimple" type="simpleFieldType"
minOccurs="0" maxOccurs="unbounded">
```

```
                <xsd:annotation>
                    <xsd:documentation>Represents simple Word
field (with plain text instructions). These fields are run-time calculated entities in Word
(for example, page numbers).</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="hlink" type="hLinkType">
                <xsd:annotation>
                    <xsd:documentation>Represents hyperlink
element (analogous to HTML &lt;a href=...&gt; tag).</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:group ref="runLevelElts"></xsd:group>
            <xsd:any processContents="skip" namespace="##other"
minOccurs="0"></xsd:any>
        </xsd:choice>
    </xsd:sequence>
    <xsd:attribute name="instr" type="stringType" use="required">
        <xsd:annotation>
            <xsd:documentation>Gets or sets instruction text for a
field.</xsd:documentation>
        </xsd:annotation>
    </xsd:attribute>
    <xsd:attribute name="fldLock" type="onOffType">
        <xsd:annotation>
            <xsd:documentation>Gets or sets whether the field is locked
from being recalculated.</xsd:documentation>
        </xsd:annotation>
    </xsd:attribute>
</xsd:complexType>

<xsd:simpleType name="fldCharTypeProperty">
    <xsd:annotation>
        <xsd:documentation>Defines a property that uses a field character
type.</xsd:documentation>
    </xsd:annotation>
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="begin"></xsd:enumeration>
        <xsd:enumeration value="separate"></xsd:enumeration>
        <xsd:enumeration value="end"></xsd:enumeration>
    </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="fldCharElt">
    <xsd:annotation>
```

```
                <xsd:documentation>Defines a field-delimiting
character.</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
                <xsd:element name="fldData" type="stringType" minOccurs="0"
maxOccurs="1">
                        <xsd:annotation>
                                <xsd:documentation>Represents field
data.</xsd:documentation>
                        </xsd:annotation>
                </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="fldCharType" type="fldCharTypeProperty">
                <xsd:annotation>
                        <xsd:documentation>Specifies whether this field-delimiting
character begins a field definition, separates the field definition from the field
instructions, or ends the field definition.</xsd:documentation>
                </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="fldLock" type="onOffType">
                <xsd:annotation>
                        <xsd:documentation>Gets or sets whether the field is locked
from being recalculated.</xsd:documentation>
                </xsd:annotation>
        </xsd:attribute>
</xsd:complexType>
```

Hyperlinks

```
<xsd:element name="hlink" type="hLinkType">
        <xsd:annotation>
                <xsd:documentation>Represents a hyperlink element (analogous to
HTML &lt;a href=...&gt; tag).</xsd:documentation>
        </xsd:annotation>
</xsd:element>

<xsd:complexType name="hLinkType">
        <xsd:annotation>
                <xsd:documentation>Defines a hyperlink in the
document.</xsd:documentation>
        </xsd:annotation>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
                <xsd:element name="r" type="rElt">
                        <xsd:annotation>
```

```
                    <xsd:documentation>Represents the run element. This is
the leaf container for data in a Word document -- text, pictures, and
so.</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="fldSimple" type="simpleFieldType"
minOccurs="0" maxOccurs="unbounded">
                <xsd:annotation>
                    <xsd:documentation>Represents simple Word field (with
plain text instructions). These fields are run-time calculated entities in Word (for
example, page numbers).</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:element name="hlink" type="hLinkType">
                <xsd:annotation>
                    <xsd:documentation>Represents the hyperlink element
(analogous to HTML &lt;a href=...&gt; tag).</xsd:documentation>
                </xsd:annotation>
            </xsd:element>
            <xsd:group ref="runLevelElts"></xsd:group>
            <xsd:any namespace="##other" minOccurs="0"
maxOccurs="unbounded" processContents="skip"></xsd:any>
        </xsd:choice>
        <xsd:attribute name="bookmark" type="stringType" use="optional">
            <xsd:annotation>
                <xsd:documentation>Specifies the bookmark location in the
document that the hyperlink will jump to.</xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="target" type="stringType" use="optional">
            <xsd:annotation>
                <xsd:documentation>Specifies the frame target for the hyperlink
(that is, the frameset).</xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="screenTip" type="stringType" use="optional">
            <xsd:annotation>
                <xsd:documentation>Specifies the text to show as a ScreenTip
for this hyperlink.</xsd:documentation>
            </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="arbLocation" type="stringType" use="optional">
            <xsd:annotation>
                <xsd:documentation>Tracks locations in documents that have no
bookmark targets. Used internally by Word.</xsd:documentation>
```

```
                </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="noHistory" type="onOffType" use="optional">
                <xsd:annotation>
                        <xsd:documentation>Gets or sets whether to add this target to the
history list when it is navigated to.</xsd:documentation>
                </xsd:annotation>
        </xsd:attribute>
        <xsd:attribute name="dest" type="stringType" use="optional">
                <xsd:annotation>
                        <xsd:documentation>Gets or sets the hyperlink
destination.</xsd:documentation>
                </xsd:annotation>
        </xsd:attribute>
</xsd:complexType>
```

Mail Merge Fields

```
<xsd:simpleType name="mailMergeDocTypeValue">
        <xsd:annotation>
                <xsd:documentation>Defines a document type for a mail-merge
operation.</xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="catalog">
                        <xsd:annotation>
                                <xsd:documentation>Catalog</xsd:documentation>
                        </xsd:annotation>
                </xsd:enumeration>
                <xsd:enumeration value="envelopes">
                        <xsd:annotation>
                                <xsd:documentation>Envelopes</xsd:documentation>
                        </xsd:annotation>
                </xsd:enumeration>
                <xsd:enumeration value="mailing-labels">
                        <xsd:annotation>
                                <xsd:documentation>Mailing
Labels</xsd:documentation>
                        </xsd:annotation>
                </xsd:enumeration>
                <xsd:enumeration value="form-letters">
                        <xsd:annotation>
                                <xsd:documentation>Form Letters</xsd:documentation>
                        </xsd:annotation>
                </xsd:enumeration>
                <xsd:enumeration value="email">
                        <xsd:annotation>
                                <xsd:documentation>E-Mail</xsd:documentation>
                        </xsd:annotation>
                </xsd:enumeration>
                <xsd:enumeration value="fax">
                        <xsd:annotation>
                                <xsd:documentation>Fax</xsd:documentation>
                        </xsd:annotation>
                </xsd:enumeration>
        </xsd:restriction>
</xsd:simpleType>
<xsd:annotation>
        <xsd:documentation> MAIL MERGE DOC TYPE PROPERTY
</xsd:documentation>
</xsd:annotation>
<xsd:complexType name="mailMergeDocTypeProperty">
        <xsd:annotation>
                <xsd:documentation>Defines a property that uses a document type for a
mail-merge operation.</xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="val" type="mailMergeDocTypeValue" use="required">
                <xsd:annotation>
                        <xsd:documentation>Gets or sets the value of a document type
for a mail-merge operation.</xsd:documentation>
                </xsd:annotation>
        </xsd:attribute>
</xsd:complexType>
<xsd:annotation>
        <xsd:documentation> MAIL MERGE DATA TYPE VALUE
</xsd:documentation>
</xsd:annotation>
<xsd:simpleType name="mailMergeDataTypeValue">
        <xsd:annotation>
                <xsd:documentation>Defines a data type for a mail-merge
operation.</xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="file"></xsd:enumeration>
                <xsd:enumeration value="Access"></xsd:enumeration>
                <xsd:enumeration value="Excel"></xsd:enumeration>
                <xsd:enumeration value="QT"></xsd:enumeration>
                <xsd:enumeration value="ODBC"></xsd:enumeration>
                <xsd:enumeration value="ODSO"></xsd:enumeration>
        </xsd:restriction>
```

```
</xsd:simpleType>
<xsd:annotation>
        <xsd:documentation> MAIL MERGE DATA TYPE PROPERTY
</xsd:documentation>
</xsd:annotation>
<xsd:complexType name="mailMergeDataTypeProperty">
        <xsd:annotation>
                <xsd:documentation>Defines a property that uses a data type for a mail-
merge operation.</xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="val" type="mailMergeDataTypeValue" use="required">
                <xsd:annotation>
                        <xsd:documentation>Gets or sets the value of a data type for a
mail-merge operation.</xsd:documentation>
                </xsd:annotation>
        </xsd:attribute>
</xsd:complexType>
<xsd:annotation>
        <xsd:documentation> MAIL MERGE DEST VALUE </xsd:documentation>
</xsd:annotation>
<xsd:simpleType name="mailMergeDestValue">
        <xsd:annotation>
                <xsd:documentation>Defines a destination for a mail-merge
operation.</xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="new-document"></xsd:enumeration>
                <xsd:enumeration value="printer"></xsd:enumeration>
                <xsd:enumeration value="email"></xsd:enumeration>
                <xsd:enumeration value="fax"></xsd:enumeration>
        </xsd:restriction>
</xsd:simpleType>
<xsd:annotation>
        <xsd:documentation> MAIL MERGE DEST PROPERTY
</xsd:documentation>
</xsd:annotation>
<xsd:complexType name="mailMergeDestProperty">
        <xsd:annotation>
                <xsd:documentation>Defines a property that uses a destination for a
mail-merge operation.</xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="val" type="mailMergeDestValue" use="required">
                <xsd:annotation>
                        <xsd:documentation>Gets or sets the value of a destination for a
mail-merge operation.</xsd:documentation>
```

```
                    </xsd:annotation>
          </xsd:attribute>
</xsd:complexType>
<xsd:annotation>
          <xsd:documentation> MAIL MERGE ODSO FMD FIELD TYPE VALUE
</xsd:documentation>
</xsd:annotation>
<xsd:simpleType name="mailMergeOdsoFMDFieldTypeValue">
          <xsd:annotation>
                    <xsd:documentation>Defines Office Data Source Object field types for a
mail-merge operation.</xsd:documentation>
          </xsd:annotation>
          <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="null"></xsd:enumeration>
                    <xsd:enumeration value="db-column"></xsd:enumeration>
                    <xsd:enumeration value="address-block"></xsd:enumeration>
                    <xsd:enumeration value="salutation"></xsd:enumeration>
                    <xsd:enumeration value="mapped"></xsd:enumeration>
                    <xsd:enumeration value="barcode"></xsd:enumeration>
          </xsd:restriction>
</xsd:simpleType>
<xsd:annotation>
          <xsd:documentation> MAIL MERGE ODSO FMD FIELD TYPE PROPERTY
</xsd:documentation>
</xsd:annotation>
<xsd:complexType name="mailMergeOdsoFMDFieldTypeProperty">
          <xsd:annotation>
                    <xsd:documentation>Defines a property that uses Office Data Source
Object field types for a mail-merge operation.</xsd:documentation>
          </xsd:annotation>
          <xsd:attribute name="val" type="mailMergeOdsoFMDFieldTypeValue"
use="required">
                    <xsd:annotation>
                              <xsd:documentation>Gets or sets the value of Office Data Source
Object field types for a mail-merge operation.</xsd:documentation>
                    </xsd:annotation>
          </xsd:attribute>
</xsd:complexType>
```

FIGURE 5 shows an exemplary flow diagram for representing field structures in a ML document, in accordance with aspects of the invention. After start block **510**, the process flows to block **520** where the fields used in a document such as a word-processor document, are determined. The fields used within a document may include many different fields, including those that are not natively supported by later applications parsing the document. Once the fields are determined processing proceeds to decision block **530**.

At decision block **530**, a determination is made whether each field used is a complex field. When the field being examined is a complex field, processing moves to block **540**. However, if the field is not a complex field, the field is a simple field and processing moves to block **550**. In another embodiment, the fields may be categorized according to fields other than complex fields and simple fields.

At block **540**, the properties of the complex field (when the field is a complex field) are mapped into elements, attributes, and values of the ML file. As an example, the fields may include "Creation Date of the Document", "Formula", a specially formatted number, a reference to text in another part of the document, or others that each have their own associated properties. Two elements used in mapping the properties of a complex field are the fldChar element and the instrText element (see FIGURE 4). The fields and the properties associated with the fields may change from page to page, section to section, chapter to chapter and the like. There may be more than one mapping, therefore, per document. Once the complex field properties are mapped, or written to the ML file, processing advances to decision block **560**.

At block **550**, the properties of the simple field (when the field is a simple field) are mapped into elements, attributes, and values. An elements used in mapping the properties of a simple field is the fldSimple element (see FIGURE 3). As previously stated, the fields and the properties associated with the fields may change from page to page, section to section, chapter to chapter and the like. There may be more than one mapping, therefore, per document. After the simple field properties are mapped, processing advances to decision block **560**.

At decision block **560**, a determination is made whether all the fields of the document have had their properties mapped to elements, attributes, and values. If not all of the fields have been processed, processing returns to block **530** where the category of the next field is determined. However, if all the fields have been processed, then the process then moves to block **570**.

At block **570**, the properties of the fields are stored in a ML document that may be read by applications that understand the ML. Once the properties are stored, processing moves to end block **580** and returns to processing other actions.

In another embodiment, the properties of each field are mapped to elements, attributes, and values without a distinction being made between complex fields and simple fields.

The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.